META-LEARNING SYMMETRIES BY REPARAMETERIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Many successful deep learning architectures are equivariant to certain transformations in order to conserve parameters and improve generalization: most famously, convolution layers are equivariant to shifts of the input. This approach only works when practitioners know the symmetries of the task and can manually construct an architecture with the corresponding equivariances. Our goal is an approach for learning equivariances from data, without needing to design custom task-specific architectures. We present a method for learning and encoding equivariances into networks by learning corresponding parameter sharing patterns from data. Our method can provably represent equivariance-inducing parameter sharing for any finite group of symmetry transformations. Our experiments suggest that it can automatically learn to encode equivariances to common transformations used in image processing tasks.

1 INTRODUCTION

In deep learning, the convolutional neural network (CNN) (LeCun et al., 1998) is a prime example of exploiting *equivariance* to a symmetry transformation to conserve parameters and improve generalization. In image classification (Russakovsky et al., 2015; Krizhevsky et al., 2012) and audio processing (Graves and Jaitly, 2014; Hannun et al., 2014) tasks, we may expect the layers of a deep network to learn feature detectors that are translation equivariant: if we translate the input, the output feature map is also translated. Convolution layers satisfy translation equivariance by definition, and produce remarkable results on these tasks. The success of convolution's "built in" inductive bias suggests that we can similarly exploit *other* equivariances to solve machine learning problems.

However, there are substantial challenges with building in inductive biases. Identifying the correct biases to build in is challenging, and even if we do know the correct biases, it is often difficult to build them into a neural network. Practitioners commonly avoid this issue by "training in" desired equivariances (usually the special case of invariances) using data augmentation. However, data augmentation can be challenging in many problem settings and we would prefer to build the equivariance into the network itself. Additionally, building in incorrect biases may actually be detrimental to final performance (Liu et al., 2018b).

In this work we aim for an approach that can automatically learn and encode equivariances into a neural network. This would free practitioners from having to design custom equivariant architectures for each task, and allow them to transfer any learned equivariances to new tasks. Neural network layers can achieve various equivariances through parameter sharing patterns, such as the spatial parameter sharing of standard convolutions. In this paper we *reparameterize* network layers to *learnably* represent sharing patterns. We leverage meta-learning to learn the sharing patterns that help a model generalize on new tasks.

The primary contribution of this paper is an approach to automatically learn equivariance-inducing parameter sharing, instead of using custom designed equivariant architectures. We show theoretically that reparameterization can represent networks equivariant to any finite symmetry group. Our experiments show that meta-learning can recover various convolutional architectures from data, and learn invariances to common data augmentation transformations.

2 Related Work

A number of works have studied designing layers with equivariances to certain transformations such as permutation, rotation, reflection, and scaling (Gens and Domingos, 2014; Cohen and Welling, 2016; Zaheer et al., 2017; Worrall et al., 2017; Cohen et al., 2019; Weiler and Cesa, 2019; Worrall and Welling, 2019). These approaches focus on manually constructing layers analagous to standard convolution, but for other symmetry groups. More theoretical work has characterized the nature of equivariant layers for various symmetry groups (Kondor and Trivedi, 2018; Shawe-Taylor, 1989; Ravanbakhsh et al., 2017). Rather than building symmetries into the architecture, data augmentation (Beymer and Poggio, 1995; Niyogi et al., 1998) trains a network to satisfy them. Diaconu and Worrall (2019) use a hybrid approach that pre-trains a basis of rotated filters in order to define roto-translation equivariant convolution. Unlike these works, we aim to automatically build in symmetries by acquiring them from data.

Prior work on automatically learning symmetries include methods for learning invariances in Gaussian processes (van der Wilk et al., 2018) and learning symmetries of physical systems (Greydanus et al., 2019; Cranmer et al., 2020). We can also consider automatic data augmentation strategies (Cubuk et al., 2018; Lorraine et al., 2019) as a way learning symmetries, though the symmetries are not embedded into the network in a transferable way.

Our work is related to neural architecture search (Zoph and Le, 2016; Brock et al., 2017; Liu et al., 2018a; Elsken et al., 2018), which also aims to automate part of the model design process. Although architecture search methods are varied, they are generally not designed to exploit symmetry or learn equivariances. Evolutionary methods for learning both network weights and topology (Stanley and Miikkulainen, 2002; Stanley et al., 2009) are also not motivated by symmetry considerations.

Our method learns to exploit symmetries that are shared by a collection of tasks, a form of meta-learning (Thrun and Pratt, 2012; Schmidhuber, 1987; Bengio et al., 1992; Hochreiter et al., 2001). We extend gradient based meta-learning (Finn et al., 2017; Li et al., 2017; Antoniou et al., 2018) to separately learn parameter sharing patterns (which enforce equivariance) and actual parameter values. Separately representing network weights in terms of a sharing pattern and parameter values is a form of *reparameterization*. Prior work has used weight reparameterization in order to "warp" the loss surface (Lee and Choi, 2018; Flennerhag et al., 2019) and to learn good latent spaces (Rusu et al., 2018) for optimization, rather than to encode equivariance. HyperNetworks (Ha et al., 2016; Schmidhuber, 1992) generate network layer weights using a separate smaller network, which can be viewed as a nonlinear reparameterization, albeit not one that encourages learning equivariances. Modular meta-learning (Alet et al., 2018) is a related technique that aims to achieve combinatorial generalization on new tasks by stacking meta-learned "modules," each of which is a neural network. This can be seen as parameter sharing by re-using and combining modules, rather than using our layerwise reparameterization.

3 Preliminaries

In Sec. 3.1, we review gradient based meta-learning, which underlies our algorithm. Sections 3.2 and 3.3 build up a formal definition of equivariance and *group convolution* (Cohen and Welling, 2016), a generalization of standard convolution which defines equivariant operations for other groups such as rotation and reflection. These concepts are important for a theoretical understanding of our work as a method for learning group convolutions in Sec. 4.2.

3.1 Gradient Based Meta-Learning

Our method is a gradient-based meta-learning algorithm that extends MAML (Finn et al., 2017), which we briefly review here. Suppose we have some task distribution $p(\mathcal{T})$, where each task dataset is split into training and validation datasets $\{\mathcal{D}_i^{tr}, \mathcal{D}_i^{val}\}$. For a model with parameters θ, loss $\mathcal{L}$, and learning rate α, the "inner loop" updates θ on the task's training data: $\theta' = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{tr})$. In the "outer loop," MAML meta-learns a good initialization θ by minimizing the loss of θ' on the task's validation data, with updates of the form $\theta \leftarrow \theta - \eta \frac{\mathrm{d}}{\mathrm{d}\theta} \mathcal{L}(\theta', \mathcal{D}^{val})$. Although MAML focuses on meta-learning the inner loop initialization θ, one can extend this idea to meta-learning other things

such as the inner learning rate α. In our method, we meta-learn a parameter sharing pattern at each layer that maximizes performance across the task distribution.

3.2 Groups and Group Actions

Symmetry and equivariance is usually studied in the context of groups and their actions on sets; refer to Dummit and Foote (2004) for more comprehensive coverage. A group G is a set closed under some associative binary operation, where there is an identity element and each element has an inverse. Consider the group $(\mathbb{Z}, +)$ (the set of integers with addition): we can add any two integers to obtain another, each integer has an additive inverse, and 0 is the additive identity.

A group G can act on a set X through some action $\rho : G \to \text{Aut}(X)$ which maps each $g \in G$ to some transformation on X. ρ must be a homomorphism, i.e. $\rho(gh) = \rho(g)\rho(h)$ for all $g, h \in G$, and $\text{Aut}(X)$ is the set of automorphisms on X (bijective homomorphisms from X to itself). As a shorthand we write $gx := \rho(g)(x)$ for any $x \in X$. Any group can act on itself by letting $X = G$: for $(\mathbb{Z}, +)$, we define the action $gx = g + x$ for any $g, x \in \mathbb{Z}$.

The action of a group G on a vector space V is called a *representation*, which we denote $\pi : G \to GL(V)$. Recall $GL(V)$ is the set of invertible linear maps on V. Assume the vectors $v \in V$ are discrete, with components $v[i]$. If we already have G's action on the indices, a natural corresponding representation is defined $(\pi(g)v)[i] := v[g^{-1}i]$. As a concrete example, consider the representation of $G = (\mathbb{Z}, +)$ for infinite length vectors. The indices are also integers, so the group is acting on itself as defined above. Then $(\pi(g)v)[i] = v[g^{-1}i] = v[i - g]$ for any $g, i \in \mathbb{Z}$. Hence this representation of $\mathbb{Z}$ *shifts* vectors by translating their indices by g spaces.

3.3 Equivariance and Convolution

A function (like a neural network layer) is equivariant to some transformation if transforming the function's input is the same as transforming its output. To be more precise, we must define what those transformations of the input and output are. Consider a neural network layer $\phi : \mathbb{R}^n \to \mathbb{R}^m$. Assume we have two representations π_1, π_2 of group G on $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively. For each $g \in G$, $\pi_1(g)$ transforms the input vectors, while $\pi_2(g)$ transforms the output vectors. The layer ϕ is G-*equivariant* with respect to these transformations if $\phi(\pi_1(g)v) = \pi_2(g)\phi(v)$, for any $g \in G, v \in \mathbb{R}^n$. If we choose $\pi_2 \equiv id$ we get $\phi(\pi_1(g)v) = \phi(v)$, showing that *invariance is a type of equivariance*.

Deep networks contain many layers, but function composition preserves equivariance. So if we achieve equivariance in each individual layer, the whole network will be equivariant. Pointwise nonlinearities such as ReLU and sigmoid are already equivariant to any permutation of the input and output indices, which includes translation, reflection, and rotation. Hence we are primarily focused on enforcing equivariance in the *linear* layers.

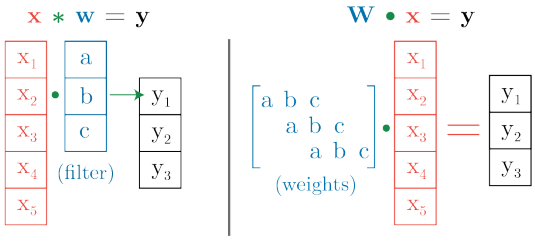

Figure 1: **Convolution as translating filters. Left**: Standard 1-D convolution slides a filter w along the length of input x. This operation is translation equivariant: translating x will translate y. **Right**: Standard convolution is equivalent to a fully connected layer with a parameter sharing pattern: each row contains translated copies of the filter. Other equivariant layers will have their own sharing patterns.

Prior work (Kondor and Trivedi, 2018) has shown that a linear layer ϕ is equivariant to the action of some group if and only if it is a group convolution, which generalizes standard convolutions to arbitrary groups. For a specific G, we call the corresponding group convolution "G-convolution" to distinguish it from standard convolution. Intuitively, G-convolution transforms a filter according to each $g \in G$, then computes a dot product between the transformed filter and the input. In standard convolution, the filter transformations correspond to translation (Fig. 1). G-equivariant layers convolve an input $v \in \mathbb{R}^n$ with a filter $\psi \in \mathbb{R}^n$. Assume the group $G = \{g_1, \cdots, g_m\}$ is finite:

$$\phi(v)[j] = (v \star \psi)[j] = \sum_i v[i](\pi(g_j)\psi)[i] = \sum_i v[i]\psi[g_j^{-1}i] \tag{1}$$

In this work, we present a method that represents and learns parameter sharing patterns for existing layers, such as fully connected layers. These sharing patterns can force the layer to implement various group convolutions, and hence equivariant layers.

4 Encoding and Learning Equivariance

To learn equivariances automatically, our method introduces a flexible representation that can encode possible equivariances, and an algorithm for learning which equivariances to encode. Here we describe this method, which we call Meta-learning Symmetries by Reparameterization (MSR).

4.1 Learnable Parameter Sharing

As Fig. 1 shows, a fully connected layer can implement standard convolution if its weight matrix is constrained with a particular *sharing pattern*, where each row contains a translated copy of the same underlying filter parameters. This idea generalizes to equivariant layers for other transformations like rotation and reflection, but the sharing pattern depends on the transformation. Since we do not know the sharing pattern a priori, we "reparameterize" fully connected weight matrices to represent them in a general and flexible fashion. A fully connected layer $\phi : \mathbb{R}^n \to \mathbb{R}^m$ with weight matrix $W \in \mathbb{R}^{m \times n}$ is defined for input x by $\phi(x) = Wx$. We can optionally incorporate biases by appending a dimension with value "1" to the input x. We factorize W as the product of a "symmetry matrix" U and a vector v of k "filter parameters":

$$\text{vec}(W) = Uv, \quad v \in \mathbb{R}^k, U \in \mathbb{R}^{mn \times k} \tag{2}$$

For fully connected layers, we reshape[1] the vector $\text{vec}(W) \in \mathbb{R}^{mn}$ into a weight matrix $W \in \mathbb{R}^{m \times n}$. Intuitively, U encodes the pattern by which the weights W will "share" the filter parameters v. Crucially, we can now separate the problem of learning the sharing pattern (learning U) from the problem of learning the filter parameters v. In Sec. 4.3, we discuss how to learn U from data.

The symmetry matrix for each layer has mnk entries, which can become too expensive in larger layers. Kronecker factorization is a common approach for approximating a very large matrix with smaller ones (Martens and Grosse, 2015; Park and Oliva, 2019). In Appendix A we describe how we apply Kronecker approximation to Eq. 2, and analyze memory and computation efficiency.

In practice, there are certain equivariances that are expensive to meta-learn, but that we know to be useful: for example, standard 2D convolutions for image data. However, there may be still other symmetries of the data (i.e., rotation, scaling, reflection, etc.) that we still wish to learn automatically. This suggests a "hybrid" approach, where we bake-in equivariances we know to be useful, and learn the others. Indeed, we can directly reparameterize a standard convolution layer by reshaping $\text{vec}(W)$ into a convolution filter bank rather than a weight matrix. By doing so we bake in translational equivariance, but we can still learn things like rotation equivariance from data.

4.2 Parameter sharing and group convolution

By properly choosing the symmetry matrix U of Eq. 2, we can force the layer to implement arbitrary group convolutions (Eq. 1) by filter v. Recall that group convolutions generalize standard convolution to define operations that are equivariant to other transformations, such as rotation. Hence by choosing U properly we can enforce various equivariances, which will be preserved regardless of the value of v.

Proposition 1 *Suppose G is a finite group $\{g_1, \ldots, g_m\}$. There exists a $U^G \in \mathbb{R}^{mn \times n}$ such that for any $v \in \mathbb{R}^n$, the layer with weights $\text{vec}(W) = U^G v$ implements G-convolution on input $x \in \mathbb{R}^n$. Moreover, with this fixed choice of U^G, any G-convolution can be represented by a weight matrix $\text{vec}(W) = U^G v$ for some $v \in \mathbb{R}^n$.*

Intuitively, U can store the symmetry transformations $\pi(g)$ for each $g \in G$, thus capturing how the filters should transform during G-convolution. For example, Fig. 2 shows how U can implement convolution on the permutation group S_2. We present a proof in Appendix B.

[1] We will use the atypical convention that vec stacks matrix entries row-wise, not column-wise.

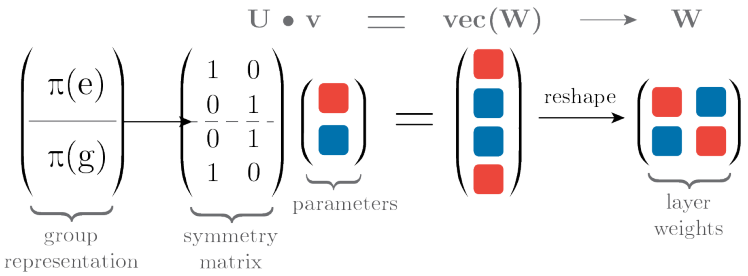

Figure 2: We reparameterize the weights of each layer in terms of a symmetry matrix U that can enforce equivariant sharing patterns of the filter parameters v. Here we show a U that enforces permutation equivariance. More technically, the layer implements group convolution on the permutation group S_2: U's block submatrices $\pi(e), \pi(g)$ define the action of each permutation on filter v. Note that U need not be binary in general.

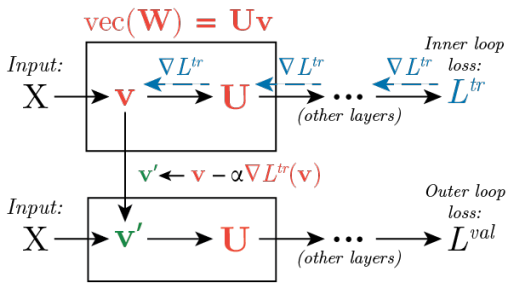

Figure 3: For each task, the inner loop updates the filter parameters v to the task using the inner loop loss. Note that the symmetry matrix U does not change in the inner loop, and is only updated by the outer loop.

Algorithm 1: MSR: Meta-Training

input: $\{\mathcal{T}_j\}_{j=1}^{N} \sim p(\mathcal{T})$: Meta-training tasks
input: $\{\mathbf{U}, \mathbf{v}\}$: Randomly initialized symmetry matrices and filters.
input: α, η: Inner and outer loop step sizes.
while *not done* **do**
  sample minibatch $\{\mathcal{T}_i\}_{i=1}^{n} \sim \{\mathcal{T}_j\}_{j=1}^{N}$;
  forall $\mathcal{T}_i \in \{\mathcal{T}_i\}_{i=1}^{n}$ **do**
    $\{\mathcal{D}_i^{tr}, \mathcal{D}_i^{val}\} \leftarrow \mathcal{T}_i$; `// task data`
    $\boldsymbol{\delta}_i \leftarrow \nabla_{\mathbf{v}} \mathcal{L}(\mathbf{U}, \mathbf{v}, \mathcal{D}_i^{tr})$;
    $\mathbf{v}' \leftarrow \mathbf{v} - \alpha \boldsymbol{\delta}_i$; `// inner step`
    `/* outer gradient */`
    $\mathbf{G}_i \leftarrow \frac{\mathrm{d}}{\mathrm{d}\mathbf{U}} \mathcal{L}(\mathbf{U}, \mathbf{v}', \mathcal{D}_i^{val})$;
  `/* outer step */`
  $\mathbf{U} \leftarrow \mathbf{U} - \eta \sum_i \mathbf{G}_\mathbf{i}$;

Subject to having a correct U^G, v is precisely the convolution filter in a G-convolution. This will motivate the notion of separately learning the convolution filter v and the symmetry structure U in the inner and outer loops of a meta-learning process, respectively.

4.3 Meta-Learning Equivariances

Meta-learning generally applies when we want to learn and exploit some shared structure in a distribution of tasks $p(\mathcal{T})$. In this case, we assume the task distribution has some common underlying symmetry: i.e., models trained for each task should satisfy some set of shared equivariances. We extend gradient based meta-learning to automatically learn those equivariances.

Suppose we have an L-layer network. We collect each layer's symmetry matrix and filter parameters: $\mathbf{U}, \mathbf{v} \leftarrow \{U^1, \cdots, U^L\}, \{v^1, \cdots, v^L\}$. Since we aim to learn equivariances that are *shared* across $p(\mathcal{T})$,the symmetry matrices should not change with the task. Hence, for any $\mathcal{T}_i \sim p(\mathcal{T})$ the inner loop fixes $\mathbf{U}$ and only updates $\mathbf{v}$ using the task training data:

$$\mathbf{v}' \leftarrow \mathbf{v} - \alpha \nabla_{\mathbf{v}} \mathcal{L}(\mathbf{U}, \mathbf{v}, \mathcal{D}_i^{tr}) \tag{3}$$

where $\mathcal{L}$ is simply the supervised learning loss, and α is the inner loop step size. During meta-training, the outer loop updates $\mathbf{U}$ by computing the loss on the task's validation data using $\mathbf{v}'$:

$$\mathbf{U} \leftarrow \mathbf{U} - \eta \frac{\mathrm{d}}{\mathrm{d}\mathbf{U}} \mathcal{L}(\mathbf{U}, \mathbf{v}', \mathcal{D}_i^{val}) \tag{4}$$

We illustrate the inner and outer loop updates in Fig. 3. Note that in addition to meta-learning the symmetry matrices, we can also still meta-learn the filter initialization $\mathbf{v}$ as in prior work. In practice we also take outer updates averaged over mini-batches of tasks, as we describe in Alg. 1.

After meta-training is complete, we freeze the symmetry matrices $\mathbf{U}$. On a new test task $\mathcal{T}_k \sim p(\mathcal{T})$, we use the inner loop (Eq. 3) to update only the filter $\mathbf{v}$. The frozen $\mathbf{U}$ enforces meta-learned

Synthetic Problems MSE (lower is better)						
	Small train dataset			Large train dataset		
Method	$k = 1$	$k = 2$	$k = 5$	$k = 1$	$k = 2$	$k = 5$
MAML-FC	$3.4 \pm .60$	$2.1 \pm .35$	$1.0 \pm .10$	$3.4 \pm .49$	$2.0 \pm .27$	$1.1 \pm .11$
MAML-LC	$2.9 \pm .53$	$1.8 \pm .24$	$.87 \pm .08$	$2.9 \pm .42$	$1.6 \pm .23$	$.89 \pm .08$
MAML-Conv	$\mathbf{.00 \pm .00}$	$.43 \pm .09$	$.41 \pm .04$	$\mathbf{.00 \pm .00}$	$.53 \pm .08$	$.49 \pm .04$
MTSR-FC (Ours)	$3.2 \pm .49$	$1.4 \pm .17$	$.86 \pm .06$	$.12 \pm .03$	$\mathbf{.07 \pm .02}$	$\mathbf{.07 \pm .01}$
MSR-FC (Ours)	$.07 \pm .02$	$\mathbf{.07 \pm .02}$	$\mathbf{.16 \pm .02}$	$\mathbf{.00 \pm .00}$	$\mathbf{.05 \pm .01}$	$\mathbf{.09 \pm .01}$

Table 1: Meta-test MSE of different methods on synthetic data with (partial) translation symmetry. "Small-/large train dataset" refers to the number of examples per training task. Among methods with non-convolutional architectures, MSR-FC is closest to matching actual convolution (MAML-Conv) performance on translation equivariant ($k = 1$) data. On data with less symmetry ($k = 2, 5$), MSR-FC outperforms MAML-Conv and other MAML approaches. MTSR is an ablation of MSR where we train the reparameterization using **m**ulti-**t**ask learning, rather than meta-learning.

parameter sharing in each layer, which improves generalization by reducing the number of task-specific inner loop parameters. For example, the sharing pattern of standard convolution makes the weight matrix constant along any diagonal, reducing the number of per-task parameters (see Fig. 1).

5 Can we recover convolutional structure?

We now introduce a series of synthetic meta-learning problems, where each problem contains regression tasks that are guaranteed to have some symmetries, such as translation, rotation, or reflection. We combine meta-learning methods with general architectures *not* designed with these symmetries in mind to see whether each method can automatically *meta-learn* these equivariances.

5.1 Learning (partial) translation symmetry

Our first batch of synthetic problems contains tasks with translational symmetry: we generate outputs by feeding random input vectors through a 1-D locally connected (LC) layer with filter size 3 and no bias. Each task corresponds to different values of the LC filter, and the meta-learner must minimize mean squared error (MSE) after observing a single input-output pair. For each problem we constrain the LC filter weights with a rank $k \in \{1, 2, 5\}$ factorization, resulting in partial translation symmetry (Elsayed et al., 2020). In the case where rank $k = 1$, the LC layer is equivalent to convolution (ignoring the biases) and thus generates exactly translation equivariant task data. We apply both MSR and MAML to this problem using a single fully connected layer (MSR-FC and MAML-FC), so these models have no translation equivariance built in and must meta-learn it to solve the tasks efficiently. For comparison, we also train convolutional and locally connected models with MAML (MAML-Conv and MAML-LC). Since MAML-Conv has built in translation equivariance, we expect it to at least perform well on the rank $k = 1$ problem. MTSR is an ablation of MSR that trains using **m**ulti-**t**ask learning instead of meta-learning. The goal is to test reparameterization (Eq. 2) *without* the meta-learning process of Alg. 1. MTSR-FC reparameterizes a fully connected layer the same way as MSR-FC. Given data from training task $\mathcal{T}_i$, MTSR jointly optimizes U (shared symmetry matrix) and $v^{(i)}$ (task specific filter parameters) using the MSE loss. For a new test task we freeze the optimized U and optimize a newly initialized filter v using the test task's training data, then evaluate MSE on held out data. Appendix D.1 further explains the experimental setup.

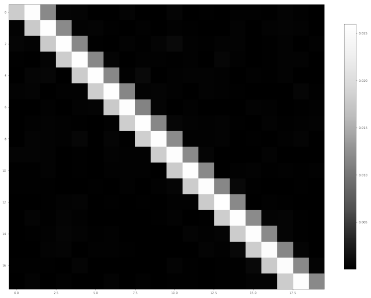

Figure 4: After observing translation equivariant data, MSR enforces convolutional parameter sharing on the weight matrix. An example weight matrix is shown above.

Table 1 shows how each method performs on each of the synthetic problems, with columns denoting the rank k of the problem's data. The "small vs large train dataset" results differ only in that the latter contains 5 or 10 *times* more examples per training task, depending on k. On fully translation equivariant data ($k = 1$), MAML-Conv performs best due to its architecture having built in translation equivariance. MSR-FC is the only non-convolutional architecture to perform comparably to MAML-Conv for $k = 1$. Fig. 4 shows that MSR-FC has learned to produce weight matrices with

convolutional parameter sharing structure, indicating it has "learned convolution" from the data. Appendix C.1 visualizes the meta-learned U, which we find implements convolution as Sec. 4.2 predicted. Meanwhile, MAML-FC and MAML-LC perform significantly worse as they are unable to meta-learn this structure. On partially symmetric data ($k = 2$, $k = 5$), MSR-FC performs well due to its ability to flexibly meta-learn even partial symmetries. MAML-Conv performs worse here since the convolution assumption is overly restrictive, while MAML-FC and MAML-LC are not able to meta-learn much structure. Across all values of k, MTSR-FC performs significantly worse than MSR-FC with small training datasets, but performs comparably with large datasets. This indicates that although our reparameterization can be trained by either multi-task learning or meta-learning, the meta-learning approach (Alg. 1) is more efficient at learning from less data.

5.2 Learning equivariance to rotations and flips

We also created synthetic problems with 2-D synthetic image inputs and outputs, in order to study rotation and flip equivariance. We generate task data by passing randomly generated inputs through a single layer E(2)-equivariant steerable CNN (Weiler and Cesa, 2019) configured to be equivariant to combinations of translations, discrete rotations by increments of 45°, and reflections. Hence our synthetic task data contains rotation and reflection in addition to translation symmetry. Each task corresponds to different values of the data-generating network's weights. We apply MSR and MAML to a single *standard* convolution layer, which guarantees translation equivariance. Each method must still meta-learn rotation and reflection (flip) equivariance from the data. Table 2 shows that MSR easily learns rotation and rotation+reflection equivariance on top of the convolutional model's built in translational equivariance. Appendix C.2 visualizes the filters MSR produces, which we see are rotated and/or flipped versions of the same filter.

Rotation/Flip Equivariance MSE		
Method	Rot	Rot+Flip
MAML-Conv	.504	.507
MSR-Conv (Ours)	**.004**	**.001**

Table 2: MSR learns rotation and flip equivariant parameter sharing on top of a standard convolution model, and thus achieves much better generalization error on meta-test tasks compared to MAML on rotation and flip equivariant data.

6 Can we learn invariances from augmented data?

Practitioners commonly use data augmentation to train their models to have certain invariances. Since invariance is a special case of equivariance, we can also view data augmentation as a way of learning equivariant models. The downside is that we need augmented data for each task. While augmentation is often possible during meta-training, there are many situations where it is impractical at meta-test time. For example, in robotics we may meta-train a robot in simulation and then deploy (meta-test) in the real world, a kind of sim2real transfer strategy (Song et al., 2020). During meta-training we can augment data using the simulated environment, but we cannot do the same at meta-test time in the real world. Can we instead use MSR to learn equivariances from data augmentation at training time, and encode those learned equivariances into the network itself? This way, the network would preserve learned equivariances on new meta-test tasks without needing any additional data augmentation.

Algorithm 2: Augmentation Meta-Training

input: $\{\mathcal{T}_i\}_{i=1}^N$: Meta-training tasks
input: META-TRAIN: Any meta-learner
input: AUGMENT: Data augmenter
forall $\mathcal{T}_i \in \{\mathcal{T}_i\}_{i=1}^N$ **do**
  $\{\mathcal{D}_i^{tr}, \mathcal{D}_i^{val}\} \leftarrow \mathcal{T}_i$; `// task data split`
  $\hat{\mathcal{D}}_i^{val} \leftarrow \text{AUGMENT}\big(\mathcal{D}^{val}\big)$;
  $\hat{\mathcal{T}}_i \leftarrow \{\mathcal{D}^{tr}, \hat{\mathcal{D}}_i^{val}\}$
META-TRAIN$\Big(\{\hat{\mathcal{T}}_i\}_{i=1}^N\Big)$

Alg. 2 describes our approach for meta-learning invariances from data augmentation, which wraps around any meta-learning algorithm using generic data augmentation procedures. Recall that each task is split into training and validation data $\mathcal{T}_i = \{\mathcal{D}_i^{tr}, \mathcal{D}_i^{val}\}$. We use the data augmentation procedure to *only modify the validation data*, producing a new validation dataset $\hat{\mathcal{D}}_i^{val}$ for each task. We re-assemble each modified task $\hat{\mathcal{T}}_i \leftarrow \{\mathcal{D}_i^{tr}, \hat{\mathcal{D}}_i^{val}\}$. So for each task, the meta-learner observes unaugmented training data, but must generalize to augmented validation data. This forces the model to be invariant to the augmentation transforms without actually seeing any augmented training data.

Method	Aug-Omniglot 5 way 1-shot	Aug-Omniglot 5 way 5-shot	Aug-Omniglot 20 way 1-shot	Aug-Omniglot 20 way 5-shot	Aug-MiniImagenet 5 way 1-shot	Aug-MiniImagenet 5 way 5-shot
MAML	87.3 ± 0.5	93.6 ± 0.3	67.0 ± 0.4	79.9 ± 0.3	42.5 ± 1.1	61.5 ± 1.0
MAML (Big)	89.3 ± 0.4	94.8 ± 0.3	69.6 ± 0.4	83.2 ± 0.3	37.2 ± 1.1	63.2 ± 1.0
ANIL	86.4 ± 0.5	93.2 ± 0.3	67.5 ± 3.5	79.8 ± 0.3	43.0 ± 1.1	62.3 ± 1.0
ProtoNets	92.9 ± 0.4	$\mathbf{97.4 \pm 0.2}$	$\mathbf{85.1 \pm 0.3}$	$\mathbf{94.3 \pm 0.2}$	34.6 ± 0.5	54.5 ± 0.6
MSR (Ours)	$\mathbf{95.3 \pm 0.3}$	$\mathbf{97.7 \pm 0.2}$	84.3 ± 0.2	92.6 ± 0.2	$\mathbf{45.5 \pm 1.1}$	$\mathbf{65.2 \pm 1.0}$

Table 3: Meta-test accuracies on Aug-Omniglot and Aug-MiniImagenet few-shot classification, which requires generalizing to augmented validation data from un-augmented training data. MSR performs comparably to or better than other methods under this augmented regime. Results shown with 95% CIs.

We apply this augmentation strategy to Omniglot (Lake et al., 2015) and MiniImagenet (Vinyals et al., 2016) few shot classification to create the *Aug-Omniglot* and *Aug-MiniImagenet* benchmarks. Our data augmentation function contains a combination of random rotations, flips, and resizes (rescaling), which we only apply to task validation data as described above. Aside from the augmentation procedure, the benchmarks are identical to prior work (Finn et al., 2017): for each task, the model must classify images into one of either 5 or 20 classes (n-way) and receives either 1 or 5 examples of each class in the task training data (k-shot).

We tried combining Alg. 2 with our MSR method and three other meta-learning algorithms: MAML (Finn et al., 2017), ANIL (Raghu et al., 2019), and Prototypical Networks (ProtoNets) (Snell et al., 2017). While the latter three methods all have the potential to learn equivariant features through Alg. 2, we hypothesize that since MSR enforces learned equivariance through its symmetry matrices it should outperform these feature-metalearning methods. Appendix D.3 describes the experimental setup and methods implementations in more detail.

Table 3 shows each method's meta-test accuracies on both benchmarks. Across different settings MSR performs either comparably to the best method, or the best. MAML and ANIL perform similarly to each other, and usually worse than MSR, suggesting that learning equivariant or invariant features is not as helpful as learning equivariant layer structures. ProtoNets perform well on the easier Aug-Omniglot benchmark, but evidently struggle with learning a transformation invariant metric space on the harder Aug-MiniImagenet problems. Note that MSR's reparameterization increases the number of meta-learned parameters at each layer, so MSR models contain more total parameters than corresponding MAML models. The "MAML (Big)" results show MAML performance with very large models containing more total parameters than the corresponding MSR models. The results show that MSR also outperforms these larger MAML models despite having fewer total parameters.

7 Discussion and Future Work

We introduce a method for automatically meta-learning equivariances in neural network models, by encoding learned equivariance-inducing parameter sharing patterns in each layer. On new tasks, these sharing patterns reduce the number of task-specific parameters and improve generalization. Our experiments show that this method can improve few-shot generalization on task distributions with shared underlying symmetries. We also introduce a strategy for meta-training invariances into networks using data augmentation, and show that it works well with our method. By encoding equivariances into the network as a parameter sharing pattern, our method has the benefit of preserving learned equivariances on new tasks so it can learn more efficiently.

Machine learning thus far has benefited from exploiting human knowledge of problem symmetries, and we believe this work presents a step towards learning and exploiting symmetries automatically. This work leads to numerous directions for future investigation. In addition to generalization benefits, standard convolution is practical since it exploits the parameter sharing structure to improve computational efficiency, relative to a fully connected layer of the same input/output dimensions. While MSR we can improve computational efficiency by reparameterizing standard convolution layers, it does not exploit learned structure to further optimize its computation. Can we automatically learn or find efficient implementations of these more structured operations? Additionally, our method is best for learning symmetries which are shared across a distribution of tasks. Further research on quickly discovering symmetries which are particular to a single task would make deep learning methods significantly more useful on many difficult real world problems.

REFERENCES

F. Alet, T. Lozano-Pérez, and L. P. Kaelbling. Modular meta-learning. *arXiv preprint arXiv:1806.10166*, 2018.

A. Antoniou, H. Edwards, and A. Storkey. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.

S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, volume 2. Univ. of Texas, 1992.

D. Beymer and T. Poggio. Face recognition from one example view. In *Proceedings of IEEE International Conference on Computer Vision*, pages 500–507. IEEE, 1995.

A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

T. Cohen and M. Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999, 2016.

T. S. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling. Gauge equivariant convolutional networks and the icosahedral cnn. *arXiv preprint arXiv:1902.04615*, 2019.

M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630*, 2020.

E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. corr abs/1805.09501 (2018). *arXiv preprint arXiv:1805.09501*, 2018.

T. Deleu, T. Würfl, M. Samiei, J. P. Cohen, and Y. Bengio. Torchmeta: A Meta-Learning library for PyTorch, 2019. URL https://arxiv.org/abs/1909.06576. Available at: https://github.com/tristandeleu/pytorch-meta.

N. Diaconu and D. E. Worrall. Learning to convolve: A generalized weight-tying approach. *arXiv preprint arXiv:1905.04663*, 2019.

D. S. Dummit and R. M. Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.

G. F. Elsayed, P. Ramachandran, J. Shlens, and S. Kornblith. Revisiting spatial invariance with low-rank local connectivity. *arXiv preprint arXiv:2002.02959*, 2020.

T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

S. Flennerhag, A. A. Rusu, R. Pascanu, H. Yin, and R. Hadsell. Meta-learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*, 2019.

R. Gens and P. M. Domingos. Deep symmetry networks. In *Advances in neural information processing systems*, pages 2537–2545, 2014.

A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pages 1764–1772, 2014.

E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, and S. Chintala. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.

S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pages 15353–15363, 2019.

D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

R. Kondor and S. Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups, 2018.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Y. Lee and S. Choi. Gradient-based meta-learning with learned layerwise metric and subspace. *arXiv preprint arXiv:1801.05558*, 2018.

Z. Li, F. Zhou, F. Chen, and H. Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018a.

R. Liu, J. Lehman, P. Molino, F. P. Such, E. Frank, A. Sergeev, and J. Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *Advances in Neural Information Processing Systems*, pages 9605–9616, 2018b.

J. Lorraine, P. Vicol, and D. Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. *arXiv preprint arXiv:1911.02590*, 2019.

J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

P. Niyogi, F. Girosi, and T. Poggio. Incorporating prior information in machine learning by creating virtual examples. *Proceedings of the IEEE*, 86(11):2196–2209, 1998.

E. Park and J. B. Oliva. Meta-curvature. In *Advances in Neural Information Processing Systems*, pages 3309–3319, 2019.

A. Raghu, M. Raghu, S. Bengio, and O. Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *arXiv preprint arXiv:1909.09157*, 2019.

S. Ravanbakhsh, J. Schneider, and B. Poczos. Equivariance through parameter-sharing. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2892–2901. JMLR. org, 2017.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

J. Schmidhuber. Evolutionary principles in self-referential learning. *On learning how to learn: The meta-meta-... hook.) Diploma thesis, Institut f. Informatik, Tech. Univ. Munich*, 1:2, 1987.

J. Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

J. Shawe-Taylor. Building symmetries into feedforward networks. In *1989 First IEE International Conference on Artificial Neural Networks,(Conf. Publ. No. 313)*, pages 158–162. IET, 1989.

J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

X. Song, Y. Yang, K. Choromanski, K. Caluwaerts, W. Gao, C. Finn, and J. Tan. Rapidly adaptable legged robots via evolutionary meta-learning. *arXiv preprint arXiv:2003.01239*, 2020.

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

M. van der Wilk, M. Bauer, S. John, and J. Hensman. Learning invariances using the marginal likelihood. In *Advances in Neural Information Processing Systems*, pages 9938–9948, 2018.

O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

M. Weiler and G. Cesa. General e (2)-equivariant steerable cnns. In *Advances in Neural Information Processing Systems*, pages 14334–14345, 2019.

D. Worrall and M. Welling. Deep scale-spaces: Equivariance over scale. In *Advances in Neural Information Processing Systems*, pages 7364–7376, 2019.

D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5028–5037, 2017.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.